BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problems Mailbox.

(19) JAPANESE PATENT OFFICE

PATENT ABSTRACTS OF JAPAN

(11) Publication number: 10260997 A

(43) Date of publication of application: 29.09.98

(51) Int. Cl G06F 17/50

(21) Application number: 09066701

(22) Date of filing: 19.03.97

(71) Applicant: TOSHIBA CORP

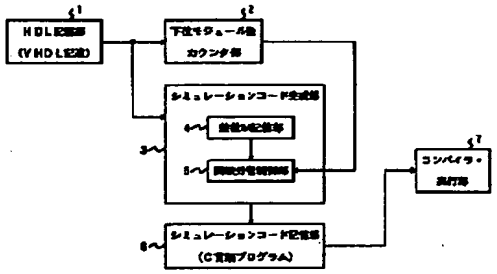
(72) Inventor: KUROSAWA YUICHI

(54) LOGIC SIMULATION SYSTEM

(57) Abstract:

PROBLEM TO BE SOLVED: To provide the logic simulation system which can suppress increases in the processing time of compilation and the generating process time of object codes and actualize simulation in an adequate processing time even if an HDL (hardware description language) description quantity increases in a process for generating a program for elaboration of a compilation type HDL simulation system.

SOLUTION: The system which performs logic simulation by converting hardware specifications to be designed which are described in VHDL into a C-language program is equipped with a low-order module number counter part 2 which counts component instances appearing in the VHDL description in the process for generating a program for elaboration and a simulation code generation part 3 which generates a function of the program each time the counter value of the counter part 2 reaches a specific reference value M.



(19)日本国特許庁（ＪＰ）　(12)公開特許公報（A）　(11)特許出願公開番号

特開平10－260997

(43)公開日　平成10年(1998)９月29日

(51)Int.Cl.⁶	識別記号	ＦＩ	
Ｇ０６Ｆ　17/50		Ｇ０６Ｆ　15/60	６６４Ｋ

審査請求　未請求　請求項の数10　ＯＬ　（全 14 頁）

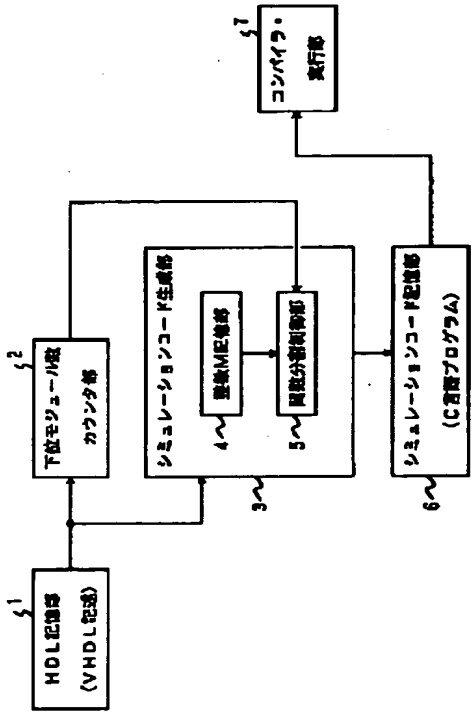
(21)出願番号　特願平9－65701

(22)出願日　平成９年(1997)３月19日

(71)出願人　000003078
株式会社東芝
神奈川県川崎市幸区堀川町72番地
(72)発明者　黒澤　雄一
東京都青梅市末広町２丁目９番地　株式会社東芝青梅工場内
(74)代理人　弁理士　鈴江　武彦　（外６名）

(54)【発明の名称】　論理シミュレーション・システム

(57)【要約】
【課題】コンパイル方式のＨＤＬシミュレーション・システムのエラボレーション用プログラムを生成する過程において、ＨＤＬ記述量が増大化した場合でも、コンパイルの処理時間及びオブジェクトコードの生成処理時間の増大化を抑制し、妥当な処理時間によるシミュレーションを実現することができる論理シミュレーション・システムを提供することにある。
【解決手段】設計対象のハードウェア仕様をＶＨＤＬにより記述された仕様記述をＣ言語プログラムに変換して、論理シミュレーションを実行するシステムにおいて、エラボレーション用プログラムを生成する過程において、ＶＨＤＬ記述中に出現するコンポーネント・インスタンスの数をカウントする下位モジュール数カウンタ部２と、このカウンタ部２のカウンタ値が所定の基準値Ｍ毎にプログラムの関数を生成するシミュレーションコード生成部３とを備えている。



1

【特許請求の範囲】

【請求項１】　設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、所定のプログラミング言語によるプログラムに変換して、このプログラムを実行する論理シミュレーション・システムであって、

前記仕様記述からデータ構造を構築するための処理過程において、前記仕様記述中に出現する下位モジュールの数をカウントする手段と、

前記下位モジュールの数のカウント値が所定の基準値以上のときに、前記基準値に相当する前記下位モジュールの数を単位として前記プログラムの関数を生成する手段とを具備したことを特徴とする論理シミュレーション・システム。

【請求項２】　設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、所定のプログラミング言語によるプログラムに変換して、このプログラムをコンピュータシステムにより実行する論理シミュレーション・システムであって、

前記所定のハードウェア記述言語により記述された仕様記述を記憶する記憶手段と、

前記仕様記述からデータ構造を前記コンピュータシステムの記憶空間上に構築するための処理過程において、前記仕様記述中に出現する下位モジュールの数をカウントするカウンタ手段と、

予め記憶した所定の基準値と前記カウンタ手段のカウンタ値とを比較して、前記カウンタ値がその基準値以上のときに、前記基準値に相当する前記下位モジュールの数を単位として前記プログラムの関数を生成し、この複数の関数からなる前記プログラムのソースコードを生成するコード生成手段とを具備したことを特徴とする論理シミュレーション・システム。

【請求項３】　前記所定のプログラミング言語はＣ言語であり、

前記Ｃ言語プログラムをコンパイルするコンパイラと、

コンパイルされたコードとライブラリとのリンク処理を実行するリンカと、

リンク処理された後のオブジェクトコードを実行することにより、前記仕様記述のシミュレーションを実行するための実行手段とを具備したことを特徴とする請求項１または請求項２記載の論理シミュレーション・システム。

【請求項４】　設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、所定のプログラミング言語によるプログラムに変換して、このプログラムをコンピュータシステムにより実行する論理シミュレーション・システムに適用するシミュレーション方法であって、

前記仕様記述からデータ構造を前記コンピュータシステムの記憶空間上に構築するための処理過程において、前

2

記仕様記述中に出現する下位モジュールの数をカウントする処理と、

予め記憶した所定の基準値と前記カウンタ手段のカウンタ値とを比較して、前記カウンタ値がその基準値以上のときに、前記基準値に相当する前記下位モジュールの数を単位として前記プログラムの関数を生成する処理と、

生成された複数の関数からなる前記プログラムのソースコードを生成する処理とからなることを特徴とするシミュレーション方法。

【請求項５】　コンピュータシステムにより読取可能な記録媒体であり、

設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、Ｃプログラミング言語によるプログラムコードに変換して、このプログラムコードにより論理シミュレーションを実行するためのプログラムを格納した記録媒体であって、

前記プログラムは、

前記仕様記述からデータ構造を前記コンピュータシステムの記憶空間上に構築するための処理過程において、前記仕様記述中に出現する下位モジュールの数をカウントする処理と、

予め記憶した所定の基準値と前記カウンタ手段のカウンタ値とを比較して、前記カウンタ値がその基準値以上のときに、前記基準値に相当する前記下位モジュールの数を単位として前記Ｃプログラミング言語における関数を生成する処理と、

生成された複数の関数からなる前記Ｃプログラミング言語のソースコードを生成する処理と、

前記ソースコードをコンパイルし、かつライブラリとのリンクする処理とを有することを特徴とする記録媒体。

【請求項６】　設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、所定のプログラミング言語によるプログラムに変換して、このプログラムを実行する論理シミュレーション・システムであって、

前記仕様記述からデータ構造を構築するための処理過程において、前記仕様記述中において宣言された信号の数をカウントする手段と、

前記信号の数のカウント値が所定の基準値以上のときに、前記基準値に相当する前記信号の数を単位として前記プログラムの関数を生成する手段とを具備したことを特徴とする論理シミュレーション・システム。

【請求項７】　設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、所定のプログラミング言語によるプログラムに変換して、このプログラムをコンピュータシステムにより実行する論理シミュレーション・システムであって、

前記所定のハードウェア記述言語により記述された仕様記述を記憶する記憶手段と、

前記仕様記述からデータ構造を前記コンピュータシステ

ムの記憶空間上に構築するための処理過程において、前記仕様記述中において宣言された信号の数をカウントするカウンタ手段と、

予め記憶した所定の基準値と前記カウンタ手段のカウンタ値とを比較して、前記カウンタ値がその基準値以上のときに、前記基準値に相当する前記信号の数を単位として前記プログラムの関数を生成し、この複数の関数からなる前記プログラムのソースコードを生成するコード生成手段とを具備したことを特徴とする論理シミュレーション・システム。

【請求項８】　前記所定のプログラミング言語はＣ言語であり、

前記Ｃ言語プログラムをコンパイルするコンパイラと、

コンパイルされたコードとライブラリとのリンク処理を実行するリンカと、

リンク処理された後のオブジェクトコードを実行することにより、前記仕様記述のシミュレーションを実行するための実行手段とを具備したことを特徴とする請求項６または請求項７記載の論理シミュレーション・システム。

【請求項９】　設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、所定のプログラミング言語によるプログラムに変換して、このプログラムをコンピュータシステムにより実行する論理シミュレーション・システムに適用するシミュレーション方法であって、

前記仕様記述からデータ構造を前記コンピュータシステムの記憶空間上に構築するための処理過程において、前記仕様記述中において宣言された信号の数をカウントする処理と、

予め記憶した所定の基準値と前記カウンタ手段のカウンタ値とを比較して、前記カウンタ値がその基準値以上のときに、前記基準値に相当する前記信号の数を単位として前記プログラムの関数を生成する処理と、

生成された複数の関数からなる前記プログラムのソースコードを生成する処理とからなることを特徴とするシミュレーション方法。

【請求項１０】　コンピュータシステムにより読取可能な記録媒体であり、

設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、Ｃプログラミング言語によるプログラムコードに変換して、このプログラムコードにより論理シミュレーションを実行するためのプログラムを格納した記録媒体であって、

前記プログラムは、

前記仕様記述からデータ構造を前記コンピュータシステムの記憶空間上に構築するための処理過程において、前記仕様記述中において宣言された信号の数をカウントする処理と、

予め記憶した所定の基準値と前記カウンタ手段のカウンタ値とを比較して、前記カウンタ値がその基準値以上のときに、前記基準値に相当する前記信号の数を単位として前記Ｃプログラミング言語における関数を生成する処理と、

生成された複数の関数からなる前記Ｃプログラミング言語のソースコードを生成する処理と、

前記ソースコードをコンパイルし、かつライブラリとのリンクする処理とを有することを特徴とする記録媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、コンピュータを使用して電子回路などのハードウェアの設計を支援するためのＣＡＤシステムに適用し、特にハードウェア記述言語により記述された論理回路設計の仕様記述の検証を行なうための論理シミュレーション・システムに関する。

【０００２】

【従来の技術】近年、ＬＳＩなどの集積回路の設計は、設計対象のハードウェア（具体的には論理回路）の複雑化及び大規模化に伴って、ハードウェアの設計仕様をハードウェア記述言語（ＨＤＬ）により記述し、ＣＡＤシステム（コンピュータシステム）を利用して論理回路の設計や設計検証を行なう設計手法が定着しつつある。このようなＨＤＬによる仕様記述を設計検証するためのシステムは、論理シミュレーション・システムまたは論理シミュレータ（ＨＤＬシミュレータ）と呼ばれている。ＨＤＬとしては、世界標準的にはＶＨＤＬ（ＶＨＳＩＣ hardware description language）またはＶｅｒｉｌｏｇＨＤＬが広く使用されている。後述する図２のＨＤＬ記述は、ＶＨＤＬ記述の具体例である。

【０００３】論理シミュレーション・システムは、通常ではＨＤＬ記述をＣ言語などのプログラミング言語からなるプログラムに変換し、このプログラムをコンパイルしてオブジェクトコードを生成し、このオブジェクトコードを実行する機能からなる。この方式はコンパイル方式と呼ばれている。

【０００４】シミュレーションのプロセスは、ＨＤＬ記述中に出現する信号、変数、モジュールなどのデータ構造をコンピュータの記憶空間上に構築する処理であるエラボレーション（ｅｌａｂｏｒａｔｉｏｎ）及びこのデータ構造をアクセスしながらＨＤＬ記述中のステートメントを実行する処理に大別される。また、Ｃ言語プログラムに変換する場合も、このプログラムをエラボレーション用プログラム及びステートメント実行用プログラムに分けて生成するのが一般的である。

【０００５】エラボレーション用プログラムは、ＨＤＬ記述の階層（モジュール）毎にプログラミング言語の１関数を生成し、またステートメント実行用プログラムは例えばＶＨＤＬやＶｅｒｉｌｏｇＨＤＬのプロセスやサブプログラム毎にプログラミング言語の１関数を生成す

るのが一般的である。

【０００６】ところで、例えばＣ言語プログラムにおいて、プログラムを構成する１関数のサイズが一定以上になった場合に、その関数を機能的に複数の関数に分割した方がコンパイルしたときのコンパイル時間を短縮できるという性質がある。ここで、１つの階層（モジュール）のＨＤＬ記述が大規模になった場合でも、ステートメント実行用プログラムは、プロセスやサブプログラム毎に１関数を生成できるため、生成される個々の関数は比較的大規模にならずにすむ。しかし、エラボレーション用プログラムは、特にゲートアレイやスタンダードセルを使用したネットリストのように信号数やサブモジュール数（ゲートアレイやスタンダードセルのインスタンス数に相当。）が多くなる場合に、生成される１関数が大きくなり、関数のコンパイルに要する時間が増大化、あるいはコンパイルそのものができなくなるという問題がある。

【０００７】図６は、従来手法において、図２に示すＶＨＤＬ記述に対するエラボレーション用コードのＣ言語による具体例を示す図である。図６において、１０～１３行目、１５～１８行目、２０～２３行目、２５～２８行目がそれぞれコンポーネント・インスタンス文Ｃ１，Ｃ２，Ｃ３，Ｃ４のエラボレーション用コード（以下エラボレーション・コードと称する）に相当する。この手法では、コンポーネント・インスタンス文Ｃ１，Ｃ２，Ｃ３，Ｃ４全てのエラボレーション・コードを同一の関数ｎｅｗ＿ｌｉｂ１＿ｅｘ１＿ｓｔｒｕｃｔｕｒｅ内に出力している。このため、ＶＨＤＬ記述中にｎ個のコンポーネント・インスタンス文が含まれる場合には、生成されるエラボレーション・コードのステートメント数は「３＊ｎ」以上になる。例えば１００００個のセルからなるゲートアレイ回路のエラボレーション・コードは、３００００ステートメント以上になる。従って、コンポーネント・インスタンス文の個数ｎが非常に大きくなると、それに伴ってエラボレーション・コードのコンパイル時間が増大し、また最悪の場合にはコンパイル不能になる可能性がある。

【０００８】

【発明が解決しようとする課題】前述したように、ＨＤＬ記述の階層（モジュール）毎に、Ｃ言語などのエラボレーション用プログラムを生成する従来方式では、１階層の記述量が増大した場合に、それに伴ってエラボレーション・コードのコンパイル時間が増大化し、オブジェクトコードの生成の処理時間が非常に増大化したり、またコンパイル自体が不可能になるという問題がある。

【０００９】そこで、本発明の目的は、いわゆるコンパイル方式のＨＤＬシミュレーション・システムのエラボレーション用プログラムを生成する過程において、ＨＤＬ記述量が増大化した場合でも、コンパイルの処理時間及びオブジェクトコードの生成処理時間の増大化を抑制し、妥当な処理時間によるシミュレーションを実現することができる論理シミュレーション・システムを提供することにある。

【００１０】

【課題を解決するための手段】本発明は、設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、所定のプログラミング言語によるプログラムに変換して、このプログラムを実行する論理シミュレーション・システムであって、仕様記述からデータ構造を構築するための処理過程において仕様記述中に出現する下位モジュールの数をカウントする手段と、下位モジュールの数のカウント値が所定の基準値以上のときに基準値に相当する前記下位モジュールの数を単位としてプログラムの関数を生成する手段とを備えたシステムである。

【００１１】即ち、本システムは、ＨＤＬ記述から例えばＣ言語プログラムを生成するときのエラボレーション用プログラムの生成過程において、１階層（１モジュール）の記述量の規模（カウント値）が大きい場合に、複数の関数に分割してプログラムを生成する方式である。従って、ＨＤＬ仕様記述に含まれる下位モジュール数がＭ個以上である場合に、Ｍ個の下位モジュールを単位としてプログラムの１関数を生成するため、１関数のサイズの増大化を抑制できる。これにより、コンパイルの処理及びオブジェクトコードの生成処理を妥当な処理時間により実行することができる。

【００１２】また、本発明は、設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、所定のプログラミング言語によるプログラムに変換して、このプログラムを実行する論理シミュレーション・システムであって、仕様記述からデータ構造を構築するための処理過程において、仕様記述中において宣言された信号の数をカウントする手段と、信号の数のカウント値が所定の基準値以上のときに、基準値に相当する信号の数を単位としてプログラムの関数を生成する手段とを備えたシステムである。

【００１３】即ち、本システムは、ＨＤＬ記述から例えばＣ言語プログラムを生成するときのエラボレーション用プログラムの生成過程において、仕様記述に含まれる信号数が多い場合に、複数の関数に分割してプログラムを生成する方式である。従って、ＨＤＬ仕様記述中に宣言された信号数がＮ個以上である場合に、Ｎ個の信号を単位としてプログラムの１関数を生成するため、１関数のサイズの増大化を抑制できる。これにより、コンパイルの処理及びオブジェクトコードの生成処理を妥当な処理時間により実行することができる。

【００１４】

【発明の実施の形態】以下図面を参照して本発明の実施の形態を説明する。

（第１の実施形態）図１は本実施形態に関係するシステ

ム構成を示すブロック図であり、図２はＶＨＤＬ記述の具体例を示す図であり、図３は本実施形態の動作を説明するためのフローチャートである。

【００１５】本実施形態は、図２に示すように、設計対象の論理回路をＶＨＤＬ記述により仕様記述を行なう場合について適用し、この仕様記述をＣ言語プログラムに変換してシミュレーションを実行するシステムを想定する。なお、ＶＨＤＬに関する文法、用語などについては、例えば「ＶＨＤＬハードウエア記述言語　日本規格協会　ＩＳＢＮ４－５４２５－４０１３６－７」などの文献に記載されている。

（システム構成）本システムは、図１に示すように、ＨＤＬ記憶部１と、下位モジュール数カウンタ部２と、シミュレーションコード生成部３と、シミュレーションコード記憶部６と、コンパイラ・実行部７とを有する。シミュレーションコード生成部３は、整数Ｍ記憶部４と関数分割制御部５とを有する。

【００１６】ＨＤＬ記憶部１は、ＶＨＤＬにより記述されたソースプログラム（以下ＶＨＤＬ記述と称する）を記憶するための要素である。ここで、ＶＨＤＬ記述は、図２に示すように、「ｅｎｔｉｔｙ　ｅｘ１」及びそのｓｔｒｕｃｔｕｒｅという名前のアーキテクチャ（ａｒｃｈｉｔｅｃｔｕｒｅ）の記述である（図２の１，５行目を参照）。

【００１７】下位モジュール数カウンタ部２は、ＨＤＬ記憶部１に記憶されたＶＨＤＬ記述においてコンポーネント・インスタンス文を１個ずつ取り出して、その数をカウントするための要素である。コンポーネント・ポーネントインスタンス文は、ＶＨＤＬにおいて下位モジュールとの接続関係を表す構文であり、例えば図２の３４，４２，５１，５８行目の文が相当する。例えば、３４行目のコンポーネント・インスタンス文は識別子がＣ１であり、６行目で宣言された下位モジュールＳＥＬ２が使用されている。

【００１８】シミュレーションコード生成部３は、前述したように、整数Ｍ記憶部４と関数分割制御部５とを有し、ＨＤＬ記憶部１に記憶されたＨＤＬ記述をシミュレーションするためのＣ言語プログラム（シミュレーションコード）を生成するための本実施形態の主要素である。整数Ｍ記憶部４は、コンポーネント・インスタンス文に対するエラボレーション・コードの出力を制御するための整数Ｍ（基準値）を記憶する。関数分割制御部５は、下位モジュール数カウンタ部２によりカウントされたコンポーネント・インスタンス数（下位モジュール数）がＭ個になる毎に、新しいＣ関数（Ｃ言語プログラムの関数）を生成し、当該コンポーネント・インスタンス文のエラボレーション・コードをそのＣ関数中に出力する処理を行なう。

【００１９】シミュレーションコード記憶部６は、シミュレーションコード生成部３により生成されたＣ言語プログラムを記憶する。なお、シミュレーションコード生成部３は、ＶＨＤＬ記述のエラボレーション・コードとステートメントを実行するためのコードとを生成する。本発明はエラボレーション・コードを生成する処理に関するものであるため、本実施形態ではエラボレーション・コードの生成処理のみについて説明する。エラボレーション・コードは、前述したように、ＶＨＤＬ記述中に出現する信号、変数、モジュールなどのデータ構造をコンピュータの記憶空間上に構築するための処理を実行するためのコードである。

（第１の実施形態のエラボレーション・コードの生成処理）本実施形態では、ＨＤＬ記憶部１には、図２に示すようなＶＨＤＬ記述が記憶されている。また、このＶＨＤＬ記述が含まれているライブラリ名は例えば「ｌｉｂ１」であると想定する。以下、図３のフローチャートを参照して、本実施形態の処理を説明する。

【００２０】まず、下位モジュールカウンタ部２がクリアされて（カウント値ｊ＝０）、関数分割制御部５の制御基準値として整数であるパラメータＭ（Ｍ＝２とする）が整数Ｍ記憶部４にセットされる（ステップＳ１）。

【００２１】ここで、１つのコンポーネント・インスタンス文に対しては、コンポーネント・インスタンス文以下の形式のＣ関数コールを生成するものとする。コンポーネント・インスタンス文以下の形式は、便宜的にスカラータイプのポートすなわち１ビット幅のポートのみからなるコンポーネントのものであると想定する。具体例としては、以下の３ステートメントを示す。

【００２２】ｈ＿ｃｏｍｐ＝ｎｅｗ＿ｖｂｌｋ０；
ｂｉｎｄｅｄａ＝ｂｉｎｄ＿ライブラリ名＿エンティティ名＿アーキテクチャ名＿コンポーネントインスタンス名（ｈ＿ｃｏｍｐ，…）；
ｓｔ＿ｃｏｍｐｏｎｅｎｔ＿ａｔｔｒ＿ｆｕｎｃ（ｈ＿ｃｏｍｐ，”コンポーネントインスタンス名”，ｂｉｎｄｅｄａ，ｈ０）；

前記１行目の関数コールはコンポーネント・インスタンスに対応するデータ構造ｈ＿ｃｏｍｐを生成する。２行目の関数コールの第２引数以降は、当該コンポーネントのポートに対応して、そのポートに接続されている信号のアドレスを実引数として設定する。ＶＨＤＬアーキテクチャ記述内のｎ番目（ｎ＝０，１，２…）に宣言された信号のアドレスは、最初の信号（図２ではＳ０）のアドレスｉｎｔｓｉｇ０を基準として、「ｉｎｔｓｉｇ０＋ｎ」とする。このように、１つのコンポーネント・インスタンス文に対して、上記３ステートメントを生成するが、これらのステートメントを全て同じ関数内に出力するのではなく、Ｍ個のコンポーネントインスタンス文毎に別のＣ関数内に出力するのが本発明の要旨である。

【００２３】シミュレーションコード生成部３は、ＨＤＬ記憶部１からＶＨＤＬ記述中のコンポーネント・イン

スタンス文Ｃｎ（ｎ＝１，２，３，４）を１つ取り出す（ステップＳ２）。ここでは、図２に示す３４行目のインスタンス文Ｃ１が得られる（ステップＳ３のＹＥＳ）。シミュレーションコード生成部３は、カウンタ部２からのカウント値ｊを入力して、パラメータＭとの比較処理を実行する。この比較処理により、カウント値ｊ（ここではｊ＝０）がＭの倍数であるため、シミュレーションコード生成部３はインスタンス文Ｃ１に対する前記３ステートメントを出力するための関数ｇを新たに生成する（ステップＳ５）。この「ｇ」の関数名は、ｎｅｗ＿ライブラリ名＿エンティティ名＿アーキテクチャ名＿ｃｏｍｐ識別子とする。識別子にはカウンタｊの値を使用する。

【００２４】そして、シミュレーションコード生成部３は、生成した関数ｇを呼び出すコードをアーキテクチャｓｔｒｕｃｔｕｒｅのエラボレーション関数内に出力する（ステップＳ６）。ここで、アーキテクチャのエラボレーション関数の名前は、ｎｅｗ＿ライブラリ名＿エンティティ名＿アーキテクチャ名とし、上位階層に対するデータ構造へのポインタｐａｒｅｎｔを引数にもつものとする。

【００２５】次に、シミュレーションコード生成部３は、インスタンス文Ｃ１に対する３ステートメントを関数ｇ、即ちｎｅｗ＿ｌｉｂ１＿ｅｘ１＿ｓｔｒｕｃｔｕｒｅ＿ｃｏｍｐ０の中に出力する（ステップＳ７）。そして、下位モジュールカウンタ部２がインクリメントして（カウント値ｊ＝１）、シミュレーションコード生成部３は、ＨＤＬ記憶部１からＶＨＤＬ記述中の次のコンポーネント・インスタンス文Ｃ２を取り出す（ステップＳ８，Ｓ２へリターン）。ここでは、図２に示す４２行目のインスタンス文Ｃ２が得られる（ステップＳ３のＹＥＳ）。シミュレーションコード生成部３は、カウンタ部２からのカウント値ｊを入力して、パラメータＭとの比較処理を実行する。この比較処理により、カウント値ｊ（ここではｊ＝１）がＭの倍数ではないため、ミュレーションコード生成部３はインスタンス文Ｃ２のエラボレーション・コードをインスタンス文Ｃ１のエラボレーション・コードの出力先と同じ関数ｇ、即ちｎｅｗ＿ｌｉｂ１＿ｅｘ１＿ｓｔｒｕｃｔｕｒｅ＿ｃｏｍｐ０の中に出力する（ステップ４のＮＯ，Ｓ７）。

【００２６】そして、下位モジュールカウンタ部２がインクリメントして（カウント値ｊ＝２）、シミュレーションコード生成部３は、ＨＤＬ記憶部１からＶＨＤＬ記述中の次のコンポーネント・インスタンス文Ｃ３を取り出す（ステップＳ８，Ｓ２へリターン）。以下同様の処理を繰り返す。即ち、図２に示す５１行目のインスタンス文Ｃ３が得られる（ステップＳ３のＹＥＳ）。ｊ（＝２）がＭ（＝２）の倍数であるため、コンポーネント・インスタンス文Ｃ１のときと同様に、ステップＳ５の処理に移行する。即ち、新しい関数ｎｅｗ＿ｌｉｂ１＿ｅｘ１＿ｓｔｒｕｃｔｕｒｅ＿ｃｏｍｐ２を生成する。さらに、これを呼び出すコードをアーキテクチャｓｔｒｕｃｔｕｒｅのエラボレーション関数ｎｅｗ＿ｌｉｂ１＿ｅｘ１＿ｓｔｒｕｃｔｕｒｅの中に出力する（ステップＳ６）。そして、ハイレベルシンセシス内にＣ３に対するエラボレーション・コードを関数ｎｅｗ＿ｌｉｂ１＿ｅｘ１＿ｓｔｒｕｃｔｕｒｅ＿ｃｏｍｐ２の中に出力し、カウンタ部２がインクリメントする（ステップＳ７，Ｓ８）。

【００２７】シミュレーションコード生成部３は、ＨＤＬ記憶部１からＶＨＤＬ記述中の次のコンポーネント・インスタンス文Ｃ４を取り出す（ステップＳ８，Ｓ２へリターン）。以下同様の処理を繰り返す。即ち、図２に示す５８行目のインスタンス文Ｃ４が得られる（ステップＳ３のＹＥＳ）。ｊ（＝３）がＭ（＝２）の倍数ではないため、コンポーネント・インスタンス文Ｃ２のときと同様に、ステップＳ７の処理に移行する。即ち、インスタンス文Ｃ４に対するエラボレーション・コードを関数ｎｅｗ＿ｌｉｂ１＿ｅｘ１＿ｓｔｒｕｃｔｕｒｅ＿ｃｏｍｐ２の中に出力し、カウンタ部２がインクリメントする（ステップＳ７，Ｓ８）次に、ＨＤＬ記憶部１には他にコンポーネント」インスタンス文は存在しないため、シミュレーションコード生成部３は処理を終了する（ステップＳ３のＮＯ）。この結果、シミュレーションコード記憶部６には、図４に示すように、生成されたアーキテクチャｓｔｒｕｃｔｕｒｅのエラボレーション・コード（一部）が格納される。また、ステップＳ４の処理で生成されたコンポーネント・インスタンスＭ（＝２）個分のエラボレーション・コードが格納される（図５を参照）。ここで、図４の関数中では、図５の関数をコールしている。図５の最初の関数ｎｅｗ＿ｌｉｂ１＿ｅｘ１＿ｓｔｒｕｃｔｕｒｅ＿ｃｏｍｐ０はコンポーネント・インスタンスＣ１，Ｃ２のエラボレーションを行うコードであり、２個目の関数ｎｅｗ＿ｌｉｂ１＿ｅｘ１＿ｓｔｒｕｃｔｕｒｅ＿ｃｏｍｐ２はコンポーネント・インスタンスＣ３，Ｃ４のエラボレーションを行うコードである。

【００２８】ところで、前述したように、図６は従来方式により、図２のＶＨＤＬ記述から生成したエラボレーション・コード（一部）を示す。図６では、コンポーネントインスタンス文Ｃ１，Ｃ２，Ｃ３，Ｃ４全てのエラボレーション・コードが同一の関数ｎｅｗ＿ｌｉｂ１＿ｅｘ１＿ｓｔｒｕｃｔｕｒｅ内に含まれる。このため、ＶＨＤＬ記述にｎ個のコンポーネント・インスタンス文が含まれる場合、生成されるエラボレーション・コードのステートメント数は３＊ｎ以上になる。このため、ｎが非常に大きくなるとエラボレーション・コードがコンパイルできなくなる可能性がある。

【００２９】これに対して、本実施形態の方式であれば、コンポーネント・インスタンス文のエラボレーション・コードを全て同一の関数内に出力するのではなく、

M個毎に異なる関数内に出力するため、コンポーネント・インスタンス文Ｃｎの数がいくつであっても、コンポーネント・インスタンス文のエラボレーション・コードを含む個々の関数のサイズは一定以下（３＊Mステートメント程度）になる。換言すれば、ＶＨＤＬ記述に含まれるコンポーネント・インスタンス文（ＨＤＬ記述の下位モジュール）をカウントし、所定値M（例えば２）毎に関数ｇ（図５の３行目と２６行目）を生成し、この関数ｇからなるＣ言語プログラムを生成する。従って、コンパイラ・実行部７が生成されたＣ言語プログラムをコンパイルするときに、各関数のサイズに制限されているため、コンパイル処理時間が極端に増大したり、またはコンパイル不能になるようなことはなく、妥当なコンパイル処理時間でオブジェクトコードを生成することが可能となる。

（第２の実施形態）図７は第２の実施形態に関係する論理シミュレーション・システムの要部を示すブロック図である。第２の実施形態のシステムは、前述の第１の実施形態において下位モジュール数カウンタ部２の代わりに信号数カウンタ部７０を有し、さらに整数M記憶部４の代わりにＮ記憶部７２を有するものである。信号数カウンタ部７０は、ＨＤＬ記憶部１に記憶されたＶＨＤＬ記述中において宣言された信号数をカウントする。また、Ｎ記憶部７２は、コンポーネント・インスタンス文に対するエラボレーション・コードの出力先に制御するためのパラメータＮ（基準値の整数）を記憶するための要素である。ここで、図２に示すＶＨＤＬ記述において、２３～３１行目のキーワード「ｓｉｇｎａｌ」が信号の宣言部分である。

【００３０】他の要素は第１の実施形態のシステムと同様の機能を有する。即ち、シミュレーションコード生成部７１は、Ｎ記憶部７２と関数分割制御部７３を有し、ＨＤＬ記憶部１に記憶されたＶＨＤＬ記述をシミュレーションするためのＣ言語プログラムを生成する。関数分割制御部７３は、信号数カウンタ部７０によりカウントされた信号数がＮ個になる毎に、新しいＣ関数を生成し、当該信号のエラボレーション・コードをそのＣ関数中に出力する。シミュレーション・コード記憶部６は、シミュレーションコード生成部７１により生成されたＣ言語プログラムを記憶する。

（第２の実施形態のエラボレーション・コードの生成処理）本実施形態においても、第１の実施形態と同様に、エラボレーション・コードの生成処理のみについて説明し、ＶＨＤＬ記述が含まれているライブラリ名は「ｌｉｂ１」であると想定する。以下、図８のフローチャートを参照して、本実施形態の処理を説明する。

【００３１】まず、信号数カウンタ部７０がクリアされて（カウント値ｉ＝０）、関数分割制御部７３の制御基準値として整数であるパラメータＮ（Ｎ＝６とする）がＮ記憶部７２にセットされる（ステップＳ１０）。

【００３２】ここで、１つの信号宣言に対しては、以下の形式のＣ関数コールを生成するものとする。以下の形式は、便宜的にスカラータイプの信号のものであると想定する。具体例としては、以下の１ステートメントを示す。

【００３３】ｓｔ＿ｓｉｇｎａｌ＿ａｔｔｒ＿ｆｕｎｃ（ｉｎｔａｉｇ０＋ｎ，ＳＣＡＬＡＬ＿ＳＩＧＮＡＬ，ＮＵＬＬ，ｈ０）；

この関数コールは、信号に対するデータ構造に属性をセットするものである。第１引数は信号のアドレスを設定する。ＶＨＤＬアーキテクチャ記述内のｎ番目（ｎ＝０，１，２…）に宣言された信号のアドレスは、最初の信号（図２ではＳ０）のアドレスｉｎｔａｉｇ０を基準として、ｉｎｔａｉｇ０＋ｎとする。第２引数は、信号がスカラータイプ（１ビットの信号）であることを示すコードである。第３引数は信号がベクトルのときのみ使用する情報であり、スカラー信号ではＮＵＬＬとする。最後の引数は、信号が定義された階層構造へのポインタをセットする。このように、１つの信号宣言に対して、上記１ステートメントを生成するが、これらのステートメントを全て同じ関数内に出力するのではなく、Ｎ個の信号毎に別のＣ関数内に出力するのが本発明の趣旨である。

【００３４】シミュレーションコード生成部７１は、ＨＤＬ記憶部１からＶＨＤＬ記述中に宣言された信号Ｓｎを１つ取り出す（ステップＳ１１）。ここでは、図２に示す２３行目のＳ０が得られる（ステップＳ１２のＹＥＳ）。シミュレーションコード生成部３は、カウンタ部７０からのカウント値ｉを入力して、パラメータＮとの比較処理を実行する。この比較処理により、カウント値Ｉ（ここではｉ＝０）がＮの倍数であるため、シミュレーションコード生成部７１は信号Ｓ０に対する前記１ステートメントを出力するための関数ｆを新たに生成する（ステップＳ１４）。この「ｆ」の関数名は、ｎｅｗ＿ライブラリ名＿エンティティ名＿アーキテクチャ名＿ｓｕｂｓｉｇ識別子とし、識別子には、カウンタｉの値を使用する。

【００３５】そして、シミュレーションコード生成部７１は、生成した関数ｆを呼び出すコードをアーキテクチャｓｔｒｕｃｔｕｒｅのエラボレーション関数内に出力する（ステップＳ１５）。ここで、アーキテクチャのエラボレーション関数は、前述の第１の実施形態施と同様である。

【００３６】次に、シミュレーションコード生成部７１は、信号Ｓ０に対するステートメントを関数ｆ、即ちｎｅｗ＿ｌｉｂ１＿ｅｘ１＿ｓｔｒｕｃｔｕｒｅ＿ｓｕｂｓｉｇ０の中に出力する（ステップＳ１６）。そして、カウンタ部７０がインクリメントして（カウント値ｉ＝１）、シミュレーションコード生成部７１は、ＨＤＬ記憶部１からＶＨＤＬ記述中の次の信号Ｓ１（２３行目）

を取り出す（ステップＳ１７，Ｓ１１へリターン）。シミュレーションコード生成部７１は、カウンタ部７０からのカウント値ｉを入力して、パラメータＮとの比較処理を実行する。この比較処理により、カウント値ｉ（ここではｉ＝１）がＮの倍数ではないため、ミュレーションコード生成部７１は信号Ｓ１のエラボレーション・コードを信号Ｓ０のエラボレーション・コードの出力先と同じ関数ｆ、即ちｎｅｗ＿ｌｉｂ１＿ｅｘ１＿ｓｔｒｕｃｔｕｒｅ＿ｓｕｂｓｉｇ０の中に出力する（ステップ１３のＮＯ，Ｓ１６）。

【００３７】そして、カウンタ部７０がインクリメントして（カウント値ｉ＝２）、シミュレーションコード生成部７１は、ＨＤＬ記憶部１からＶＨＤＬ記述中の次の信号を取り出す（ステップＳ１７，Ｓ１１へリターン）。以下同様の処理を繰り返す。即ち、ステップＳ１１では、信号Ｄ０，Ｄ１，Ｄ２，Ｄ３がそれぞれ、カウント値ｉ＝２，３，４，５のときに取り出される。これらの場合、ｉがＮの倍数でないため、前述の信号Ｓ１の場合と同様に、そのエラボレーション・コードは関数ｎｅｗ＿ｌｉｂ１＿ｅｘ１＿ｓｔｒｕｃｔｕｒｅ＿ｓｕｂｓｉｇ０の中に出力される（ステップＳ１６）。

【００３８】次に、カウンタ部７０がインクリメントして（カウント値ｉ＝６）、シミュレーションコード生成部７１は、ＨＤＬ記憶部１からＶＨＤＬ記述中の次の信号Ｃを取り出す（ステップＳ１７，Ｓ１１へリターン）。ここで、ｉがＮの倍数であるため、ステップＳ１４の処理に移行する。即ち、シミュレーションコード生成部７１は、新しい関数ｎｅｗ＿ｌｉｂ１＿ｅｘ１＿ｓｔｒｕｃｔｕｒｅ＿ｓｕｂｓｉｇ６を生成し、この関数をコールするコードをアーキテクチャエラボレーション関数ｎｅｗ＿ｌｉｂ１＿ｅｘ１＿ｓｔｒｕｃｔｕｒｅ内に出力する（ステップＳ１５）。以下、ステップＳ１７においてｉ＝７，８，９となったときに、ステップＳ１１において、それぞれ信号ＣＬＫ，Ｑ１，Ｑ２が取り出されてゆくが、ｉがＮの倍数でないため、これらのエラボレーション・コードは、関数ｎｅｗ＿ｌｉｂ１＿ｅｘ１＿ｓｔｒｕｃｔｕｒｅ＿ｓｕｂｓｉｇ６中に出力される（ステップＳ１６）。そして、ステップＳ１１において、これ以上信号が得られないため、ステップＳ１２がＮＯとなって処理が終了する。

【００３９】この結果、シミュレーションコード記憶部６には、図９に示すように、生成されたアーキテクチャｓｔｒｕｃｔｕｒｅのエラボレーション・コード（一部）が格納される。また、ステップＳ１４の処理で生成された信号Ｎ（＝６）個分のエラボレーション・コードが格納される（図１０を参照）。ここで、図９の関数中の９，１０行目では、図１０の関数がコールされている。図１０において、最初の関数内の７～１２行目は、図２の信号Ｓ０～Ｄ３に対するエラボレーション・コードであり、２個目の関数内の２２～２５行目は信号Ｃ～Ｑ１に対するエラボレーション・コードである。

【００４０】ところで、前述したように、図１１は従来方式により、図２のＶＨＤＬ記述から生成したエラボレーション・コード（一部）を示す。図１１では、図２の記述中で宣言されたＳ０～Ｑ１全ての信号１０個のエラボレーション・コードが同一の関数ｎｅｗ＿ｌｉｂ１＿ｅｘ１＿ｓｔｒｕｃｔｕｒｅ内に含まれる。このため、ＶＨＤＬ記述にｎ個の信号宣言が含まれる場合、生成されるエラボレーション・コードのステートメント数はｎ以上になる。このため、ｎが非常に大きくなるとエラボレーション・コードがコンパイルできなくなる可能性がある。

【００４１】これに対して、本実施形態の方式であれば、信号のエラボレーション・コードを全て同一の関数内に出力するのではなく、Ｎ個ずつ異なる関数内に出力するため、信号数がいくつであっても、信号のエラボレーション・コードを含む個々の関数のサイズは一定以下（Ｎステートメント程度）になる。従って、生成されたＣ言語プログラムのコンパイル処理時間を短縮することができる。また、ＶＨＤＬ記述に含まれる信号数がいくつになってもコンパイルができなくなるということはない。従って、コンパイラ・実行部７が生成されたＣ言語プログラムをコンパイルするときに、各関数のサイズに制限されているため、コンパイル処理時間が極端に増大したり、またはコンパイル不能になるようなことはなく、妥当なコンパイル処理時間でオブジェクトコードを生成することが可能となる。

（第３の実施形態）第３の実施形態は、前述の第１と第２の実施形態の組み合わせたものである。即ち、第１の実施形態は、下位モジュール（ＶＨＤＬ記述ではコンポーネント・インスタンス文により記述）数の多いＨＤＬ記述に対して有効であり、第２の実施形態は信号数の多いＨＤＬ記述に対して有効である。

【００４２】図１２は第３の実施形態に関係するシステムのブロック図である。本システムにおいて、ＨＤＬ記憶部１は、第１と第２の実施形態と同様に、ＶＨＤＬ記述を記憶する。下位モジュール数カウンタ部２は、第１の実施形態と同様に、ＨＤＬ記憶部１に記憶されたＶＨＤＬ記述においてコンポーネント・インスタンス文を１個ずつ取り出しその数をカウントする。信号数カウンタ部７０は、第２の実施形態７０と同様に、ＨＤＬ記憶部１に記憶されたＶＨＤＬ記述において宣言された信号を１個ずつ取り出し、その数をカウントする。

【００４３】さらに、シミュレーションコード生成部８０は、Ｍ記憶部４と、Ｎ記憶部７２と、関数分割制御部８１とを有し、ＨＤＬ記憶部１に記憶されたＶＨＤＬ記述をシミュレーションするためのＣ言語プログラムを生成する。Ｍ記憶部４は、第１の実施形態と同様に、コンポーネント・インスタンス文に対するエラボレーション・コードの出力先を制御するためのパラメータＭを記憶

する。Ｎ記憶部７２は、第２の実施形態と同様に、信号に対するエラボレーション・コードの出力先を制御するためのパラメータＮを記憶する。関数分割制御部８１は、信号に対するエラボレーション・コードを生成する場合においては、図８のフローチャートで示す同様の処理を実行し、信号数カウンタ部７０によりカウントされた信号数がＮ個になる毎に新しいＣ関数を生成し、当該信号のエラボレーション・コードをそのＣ関数中に出力する。また、関数分割制御部８１は、コンポーネント・インスタンス文に対するエラボレーション・コードを生成する場合においては、図３のフローチャートで示す同様の処理を実行し、下位モジュールカウンタ部２によりカウントされたコンポーネント・インスタンス数がＭ個になる毎に、新しいＣ関数を生成し、当該コンポーネント・インスタンスのエラボレーション・コードをそのＣ関数中に出力する。シミュレーションコード記憶部６は、シミュレーションコード生成部８０により生成されたＣ言語プログラムを記憶する。

【００４４】ここで、Ｍ＝２，Ｎ＝６として、図２のＶＨＤＬ記述から、シミュレーションコード生成部８０により生成されたエラボレーション・コードの一部を図１３に示す。図１３において、９，１０行目は信号のエラボレーション・コードをコールする部分であり、１５，１６行目はコンポーネント・インスタンスのエラボレーション・コードをコールする部分である。コールされた信号Ｎ個分のエラボレーション・コード及びコンポーネント・インスタンスＭ個分のエラボレーション・コードは、それぞれ図１０、図５と同様である。

（第３の実施形態の変形例１）第３の実施形態において、シミュレーションコード記憶部６に記憶される関数をそれぞれ別のファイルに記憶してもよい。別ファイルに記憶し、ファイル単位でコンパイルをかけることによって、全ての関数を１ファイルに出力する場合と比較して処理時間を短縮化できる可能性がある。

（第３の実施形態の変形例２）第３の実施形態において、生成されるエラボレーション・コードの量を制御するパラメータＭ，Ｎの設定を固定とせずに、本システムが実行される前に予め外部から設定できるようにしてもよい。例えば、パラメータＭ，Ｎの値をファイルから読み込むことが考えられる。あるいは、本システムが実行されるＯＳ（例えば、ＵＮＩＸ）の環境変数として定義し、設定してもよい。

（第１から第３の実施形態の変形例１）第１、第２、第３の各実施形態においては、ＨＤＬ記述をＣ言語プログラムに変換してシミュレーションを行う方式の場合を説明したが、他のプログラミング言語に変換してシミュレーションを行う方式の場合も本発明を適用することができる。また、Ｃ言語などの高級プログラミング言語を介さずに、アセンブラ言語のコードを直接生成して、シミュレーションを行う方式の場合にも適用することができる。アセンブラ言語のコードを生成する場合も、高級プログラミング言語の場合と同様に、エラボレーション・コードを適当なサイズに関数化して出力することによって、オブジェクトコードの生成に要する処理時間を短縮化することができる。

（第１から第３の実施形態の変形例２）第１、第２、第３の各実施形態においては、ＨＤＬ記述としてＶＨＤＬの場合を示したが、ＶｅｒｉｌｏｇＨＤＬなど他のＨＤＬ記述の場合も適用することができる。

（第１から第３の実施形態の変形例３）第１、第２、第３の各実施形態において、エラボレーション・コードの生成処理及びコンパイラの処理を含むソフトウェアを、例えばＣＤ－ＲＯＭなどの光ディスクやフロッピーディスクなどの磁気ディスクに格納した記録媒体も本発明の適用範囲である。この記録媒体を、パーソナルコンピュータやワークステーションなどのコンピュータシステムにセットすることにより、前述した第１、第２、第３の各実施形態における論理シミュレーションを実行することができる。

【００４５】

【発明の効果】以上詳述したように本発明によれば、コンパイル方式のＨＤＬシミュレーション・システムにおいて、特にエラボレーション用プログラムを生成する過程において、ＨＤＬ記述量が増大化した場合でも、コンパイルの処理時間及びオブジェクトコードの生成処理時間の増大化を抑制することができる。従って、大規模なＨＤＬ記述に対しても妥当な処理時間によるシミュレーションを実現することができる。

【図面の簡単な説明】

【図１】本発明の第１の実施形態に関係する論理シミュレーション・システムの要部を示すブロック図。

【図２】本実施形態に関係するＶＨＤＬ記述の具体例を示す図。

【図３】本実施形態の動作を説明するためのフローチャート。

【図４】本実施形態に関係するエラボレーション・コードを示す図。

【図５】本実施形態に関係するエラボレーション・コードを示す図。

【図６】従来のＶＨＤＬ記述に対するエラボレーション用コードのＣ言語による具体例を示す図。

【図７】第２の実施形態に関係する論理シミュレーション・システムの要部を示すブロック図。

【図８】第２の実施形態の動作を説明するためのフローチャート。

【図９】第２の実施形態に関係するエラボレーション・コードを示す図。

【図１０】第２の実施形態に関係するエラボレーション・コードを示す図。

【図１１】従来方式におけるエラボレーション・コード

を示す図。

【図１２】第３の実施形態に関係する論理シミュレーション・システムの要部を示すブロック図。

【図１３】第３の実施形態に関係するエラボレーション・コードを示す図。

【符号の説明】

１…ＨＤＬ記憶部

２…下位モジュール数カウンタ部

３…シミュレーションコード生成部

４…整数Ｍ記憶部

５…関数分割制御部

６…シミュレーションコード記憶部

７…コンパイラ・実行部

７０…信号数カウンタ部

７１…シミュレーションコード生成部

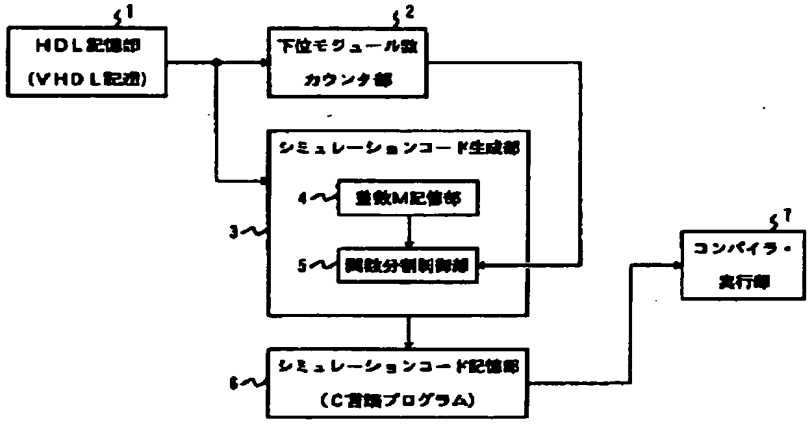
７２…Ｎ記憶部

７３…関数分割制御部

８０…シミュレーションコード生成部

８１…関数分割制御部

【図１】



【図４】

```
行
 4  VBLKP new_libl_ex1_structure(parent)
 5  VBLKP parent;
    {
        途中省略。

10      new_libl_ex1_structure_comp0(h0, intsig0);

12      new_libl_ex1_structure_comp2(h0, intsig0);

        return(h0);
    }
```

【図２】

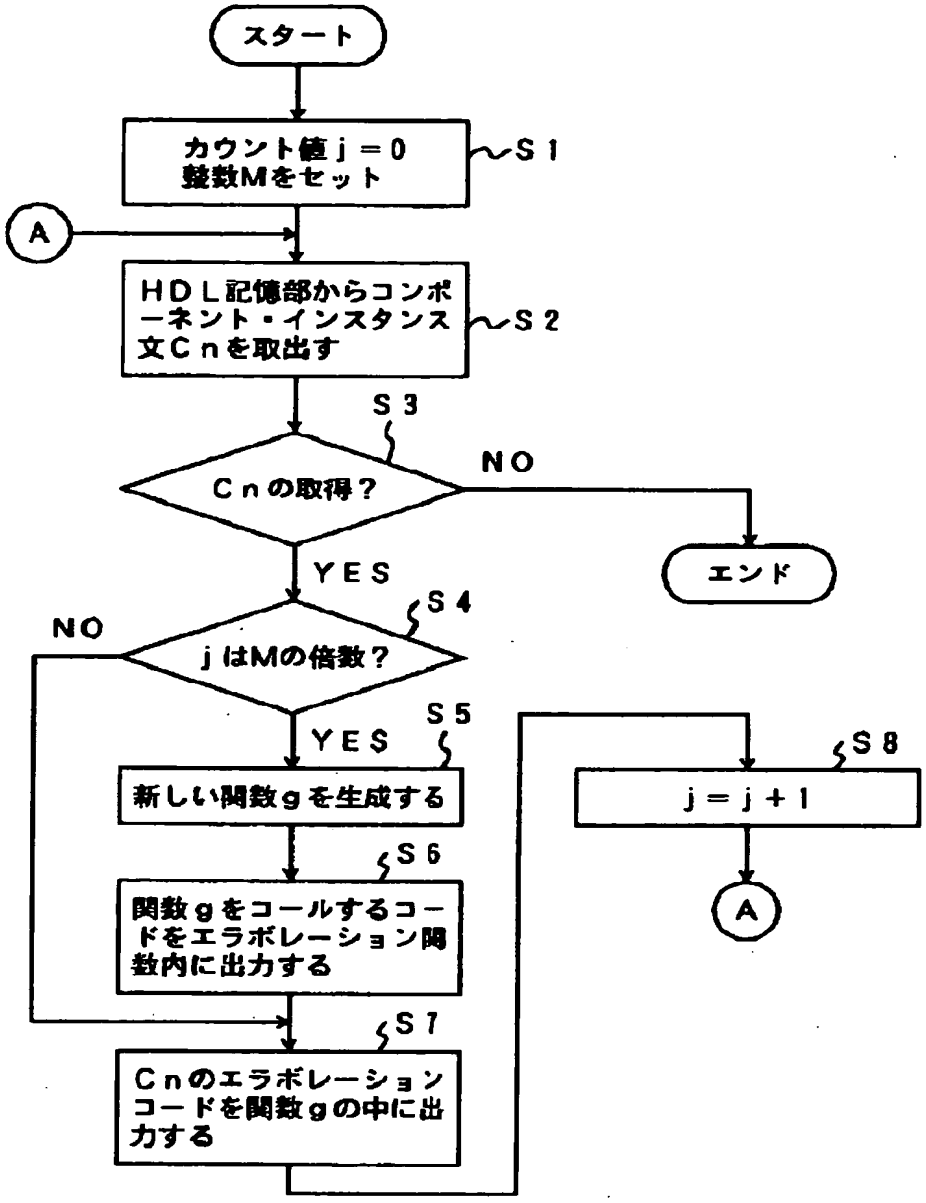
```
行
 1  entity ex1 is
 2  end EX1;

 5  architecture structure of EX1 is
      component SEL2
      port(
        I0 : in bit;
        I1 : in bit;
        C  : in bit;
        O0 : out bit;
      );
      end component;

      component DFF
      port(
        D  : in bit;
        CK : in bit;
        Q  : out bit;
        );
      end component;

23      signal S0,S1 : bit;
        signal  D0 :  bit;
        signal  D1 :  bit;
        signal  D2 :  bit;
        signal  D3 :  bit;
        signal  C :   bit;
        signal  CLK : bit;
        signal  Q0 :  bit;
31      signal  Q1 :  bit;
```

```
行
    begin
34    C1 : SEL2
      port map(
        I0 => D0,
        I1 => D1,
        C  => C,
        O0 => S0,
        );

42    C2 : SEL2
      port map(
        I0 => D2,
        I1 => D3,
        C  => C,
        O0 => S1,
        );

51    C3 : DFF
      port map(
        D => S0,
        CK => CLK,
        Q => Q0
        );

58    C4 : DFF
      port map(
        D => S1,
        CK => CLK,
        Q => Q1
        );

    end structure;
```

【図６】

```
行
 4  VBLKP new_libl_ex1_structure(parent)
 5  VBLKP parent;
    {
        途中省略。

10  h_comp = new_vblk();
    bindeda = bind_libl_ex1_structure_sel2(h_comp,
            intsig0+2, intsig0+3, intsig0+6, intsig0+0 ) ;
13  st_component_attr_func(h_comp, "c1",bindeda, h0) ;

15  h_comp = new_vblk();
    bindeda = bind_libl_ex1_structure_sel2(h_comp,
            intsig0+4, intsig0+5, intsig0+6, intsig0+1 ) ;
    st_component_attr_func(h_comp, "c2",bindeda, h0) ;

20  h_comp = new_vblk();
    bindeda = bind_libl_ex1_structure_dff(h_comp,
                                intsig0+0, intsig0+7, intsig0+3);
    st_component_attr_func(h_comp, "c3",bindeda, h0) ;

25  h_comp = new_vblk();
    bindeda = bind_libl_ex1_structure_dff(h_comp,
                                intsig0+1, intsig0+7, intsig0+9);
    st_component_attr_func(h_comp, "c4",bindeda, h0) ;

    return(h0);

    }
```

【図３】


スタート
カウント値ｊ＝０
整数Ｍをセット
S１
A
ＨＤＬ記憶部からコンポーネント・インスタンス文Ｃｎを取出す
S２
S３
Ｃｎの取得？
NO
YES
エンド
S４
NO
ｊはＭの倍数？
S５
YES
新しい関数ｇを生成する
S６
関数ｇをコールするコードをエラボレーション関数内に出力する
S７
Ｃｎのエラボレーションコードを関数ｇの中に出力する
S８
ｊ＝ｊ＋１
A

【図５】

```
行
 3  void new_libl_exl_structure_comp0(h0, intsig0)
    VBLKP h0;
    SignalP intsig0;
    {
        VBLKP h_comp;
        VBLKP bindeda;
        extern VBLKP bind_libl_exl_structure_sel2();

11      h_comp = new_vblk();
        bindeda = bind_libl_exl_structure_sel2(
                h_comp.newpathidx,
                intsig0+2, intsig0+3, intsig0+6, intsig0+0 ) ;
        st_component_attr_func(h_comp, "c1", bindeda, h0) ;

17      h_comp = new_vblk();
        bindeda = bind_libl_exl_structure_sel2(
                h_comp.newpathidx,
                intsig0+4, intsig0+5, intsig0+6, intsig0+1 ) ;
        st_component_attr_func(h_comp, "c2", bindeda, h0) ;
    }

26  void new_libl_exl_structure_comp2(h0, intsig0)
    VBLKP h0;
    SignalP intsig0;
    {
        VBLKP h_comp;
        VBLKP bindeda;
        extern VBLKP bind_libl_exl_structure_dff();

34      h_comp = new_vblk();
        bindeda = bind_libl_exl_structure_dff(h_comp,
                                intsig0+0, intsig0+7, intsig0+8) ;
        st_component_attr_func(h_comp, "c3", bindeda, h0) ;

39      h_comp = new_vblk();
        bindeda = bind_libl_exl_structure_dff(h_comp,
                                intsig0+1, intsig0+7, intsig0+9) ;
        st_component_attr_func(h_comp, "c4", bindeda, h0) ;
    }
```

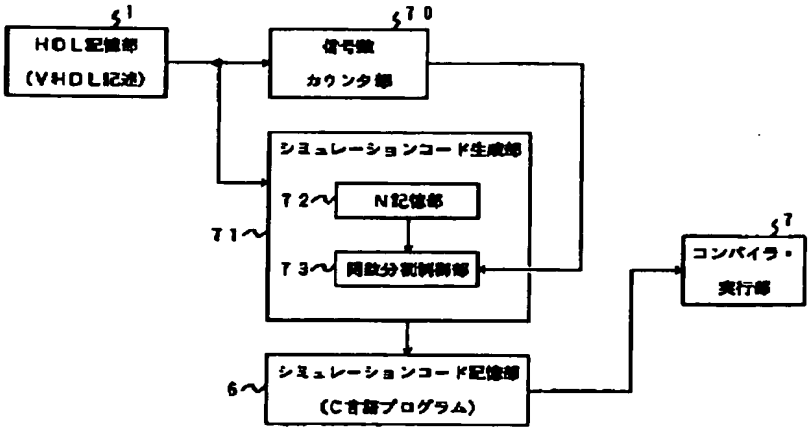
【図９】

```
行
 4  VBLKP new_libl_exl_structure(parent)
 5  VBLKP parent;
    {
        途中略

 9      new_libl_exl_structure_subsig0(intsig0, h0);
10      new_libl_exl_structure_subsig6(intsig0, h0);

12      h_comp = new_vblk();
13      bindeda = bind_libl_exl_structure_sel2(h_comp,
                        intsig0+2, intsig0+3, intsig0+6, intsig0+0);

15      st_component_attr_func(h_comp, "c1", bindeda, h0);

        途中略

        return(h0);
    }
```

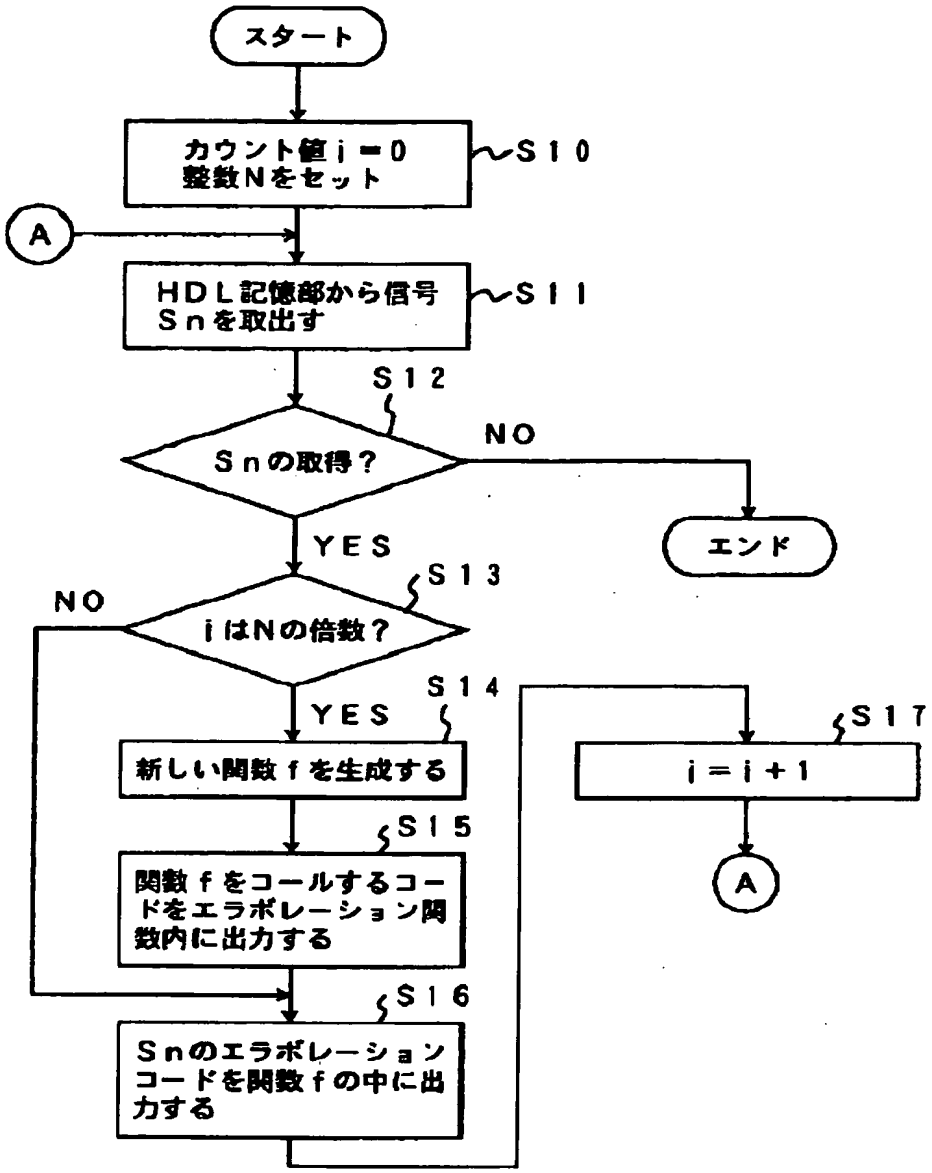
【図１３】

```
行
 4  VBLKP new_libl_exl_structure(parent)
 5  VBLKP parent;
    {
        途中略

 9      new_libl_exl_structure_subsig0(intsig0, h0);
10      new_libl_exl_structure_subsig6(intsig0, h0);

        途中略

15      new_libl_exl_structure_comp0(h0, intsig0);
16      new_libl_exl_structure_comp2(h0, intsig0);

        return(h0);
    }
```

【図７】



【図８】


スタート
カウント値ｉ＝０
整数Ｎをセット
S１０
A
HDL記憶部から信号
Ｓｎを取出す
S１１
S１２
Ｓｎの取得？
NO
エンド
YES
S１３
NO
ｉはＮの倍数？
S１４
YES
新しい関数ｆを生成する
S１５
関数ｆをコールするコードをエラボレーション関数内に出力する
S１６
Ｓｎのエラボレーションコードを関数ｆの中に出力する
S１７
ｉ＝ｉ＋１
A

【図１０】

```
行
3  void new_dis_kurosawa_ex1_structure_subsig0(intsig0, h0)
   SignalP intsig0;
   VBLKP h0;
   {
7      st_signal_attr_func(intsig0+0, SCALAR_SIGNAL, NULL, h0);
       st_signal_attr_func(intsig0+1, SCALAR_SIGNAL, NULL, h0);
       st_signal_attr_func(intsig0+2, SCALAR_SIGNAL, NULL, h0);
       st_signal_attr_func(intsig0+3, SCALAR_SIGNAL, NULL, h0);
       st_signal_attr_func(intsig0+4, SCALAR_SIGNAL, NULL, h0);
       st_signal_attr_func(intsig0+5, SCALAR_SIGNAL, NULL, h0);
   }


18 void new_dis_kurosawa_ex1_structure_subsig6(intsig0, h0)
   SignalP intsig0;
   VBLKP h0;
   {
22     st_signal_attr_func(intsig0+6, SCALAR_SIGNAL, NULL, h0);
       st_signal_attr_func(intsig0+7, SCALAR_SIGNAL, NULL, h0);
       st_signal_attr_func(intsig0+8, SCALAR_SIGNAL, NULL, h0);
       st_signal_attr_func(intsig0+9, SCALAR_SIGNAL, NULL, h0);

   }
```

【図１１】

```
行
4  VBLKP new_dis_kurosawa_ex1_structure(parent)
   VBLKP parent;
   int pathidx;
   {
       途中略。

10     st_signal_attr_func(intsig0+0, SCALAR_SIGNAL, NULL, h0);
       st_signal_attr_func(intsig0+1, SCALAR_SIGNAL, NULL, h0);
       st_signal_attr_func(intsig0+2, SCALAR_SIGNAL, NULL, h0);
       st_signal_attr_func(intsig0+3, SCALAR_SIGNAL, NULL, h0);
       st_signal_attr_func(intsig0+4, SCALAR_SIGNAL, NULL, h0);
       st_signal_attr_func(intsig0+5, SCALAR_SIGNAL, NULL, h0);
       st_signal_attr_func(intsig0+6, SCALAR_SIGNAL, NULL, h0);
       st_signal_attr_func(intsig0+7, SCALAR_SIGNAL, NULL, h0);
       st_signal_attr_func(intsig0+8, SCALAR_SIGNAL, NULL, h0);
       st_signal_attr_func(intsig0+9, SCALAR_SIGNAL, NULL, h0);


22     h_comp = new_vblk();
23     bindada = bind_lib1_ex1_structure_sel2(h_comp,
                                intsig0+2, intsig0+3, intsig0+6, intsig0+0);

       st_component_attr_func(h_comp, "c1", bindada, h0);

        途中略

       return(h0);
   }
```

【図１２】